自然言語処理 63－7
（1987. 9. 25）

プログラム仕様に用いる自然語の形式的意味定義について

並河 英二　松村 享　関 浩之
藤井 譲　嵩 忠雄
大阪大学基礎工学部情報工学科

プログラム仕様の記述に有用と思われる英語の部分言語L_{NS}の構文と意味を代数的言語ＡＳＬ／＊を用いて定義している．L_{NS}の構文は，文脈自由文法で定義し，L_{NS}の意味は，論理式への変換則を定義する公理と，個々の語句を定義する公理によって定義する．仕様記述に用いる自然語を対象としていることを考慮し，個々の語句は，抽象的データタイプの上の演算や述語として定義する．L_{NS}の構文は，文脈自由文法の記述法の一つであるＧＰＳＧを用いて定義した．特に，データタイプに関する構文情報を非終端記号の引数として表現し，構文を精密化した．変換則については，我々の当初の定義を拡張し，複数形名詞句，いわゆるdonkey文，及び等位接続に関する形式的定義を与えた．

On a Formal Definition of a Fragment of English Used for Writing Program Specifications

Eiji NABIKA　Takashi MATSUMURA　Hiroyuki SEKI
Mamoru FUJII　Tadao KASAMI

Department of Information and Computer Sciences
Faculty of Engineering Science, Osaka University
Toyonaka, Osaka, 560 JAPAN

A 'fragment' of English which is used for writing program specifications is defined by algebraic language ASL/*. The syntax of L_{NS} is defined by giving a generalized phrase structure grammar and the semantics of L_{NS} is defined by giving algebraic axioms which define the transformation rules from sentences in L_{NS} into formulas, and axioms which define the semantics of 'non-logical' words as operations or predicates on abstract data types. To refine the syntax, for each phrase, the data types of the phrase and its arguments are specified as features (arguments) of non-terminal symbols. We also describe the transformation rules which define, among others, the semantics of plural noun phrases, so called 'donkey sentences' and coordinations.

１．ま え が き

プログラムの段階的詳細化において，プログラムが仕様を満足していることの形式的な検証が行えるためには，仕様やプログラムの意味が形式的に定義されていることが望ましい。一方，書き手の意図をできるだけ正確に記述に反映し，かつ，記述の意味するところを直観的にわかり易いものにするため，通常，プログラム仕様は日常用いる自然語によって記述される。

筆者らは，既に，プログラム仕様の記述に有用と思われるものに限定した自然語の部分言語（L_{NS}とかく）の意味定義法について検討してきた。文献(8)で述べた理由により，L_{NS}の構文及び意味を定義するための言語としては，代数的言語ＡＳＬ／＊[5]を選択した。また，L_{NS}として英語の部分言語を例にとり，実際に意味定義を行う作業を進めている。さらに，これらの意味定義に基づき，L_{NS}で記述された仕様を形式的な仕様に変換するための支援システムを作成中である[4]。

２．で述べるように，我々の手法では，まず，L_{NS}の構文を指定する規則（構文則）を与え，次に，L_{NS}に属する各文を，適当な論理体系のもとでの論理式に変換する規則（変換則）を定める。また，個々の語句（本稿では非論理的要素と呼ぶ）の意味は，変換則とは別に与える。意味定義の対象をプログラム仕様に限定しているので，個々の語句は，いわゆる抽象的データタイプ上の演算や述語として定義するという方法をとる。これにより，仕様を段階的に詳細化してプログラムを導出する過程を，抽象的データタイプに基づく記述の詳細化という形式的な枠組みでとらえることができ，プログラムの正当性の検証等に関する議論が見通しよく行える。さて，筆者らが文献(7)(8)で示した定義法では，L_{NS}の構文については，句構造文法による標準的な自然語の構文指定法に従ってＧＰＳＧ（文脈自由文法の一つの記述法[2]）を用いて定義し，個々の語句がどのような抽象的データタイプ上の演算であるかといった情報は，ＧＰＳＧによる記述とは別に与えていた。しかし，このように構文の定義が“二重構造”をもつことは，定義の簡潔さを保つ上で望ましくなく，例えば，構文解析時のあいまいさを少なくするためには，ＧＰＳＧによる構文記述に基づく構文解析だけでは不十分である。そこで，対策として，データタイプに関する情報をＧＰＳＧの非終端記号の引数として表現することにより，L_{NS}の構文を，データタイプに関する構文情報も含めてＧＰＳＧで定義する方法をとる（３．参照）．４．では，当初は考慮していなかった(a)複数形名詞句，(b)いわゆるdonkey文，及び，(c)等位接続された名詞句に関する定義を中心に，変換則の概要を述べる。５．では，３．及び４．の定義をもとに例文を解析する。

２．意味定義の概要

L_{NS}に属する各文の意味を定義するため，L_{NS}の各文を適当な論理体系の論理式に対応づける規則（変換則とよぶ）を定義する。変換則によってL_{NS}の文ｓが論理式ｓ’に対応づけられるとき，ｓの意味をその論理体系におけるｓ’の意味に等しいと定義する。論理式への変換による意味定義の方法として，Montagueらの意味定義法[1]を参考にした。ただし，Montagueらは，論理体系として内包論理を採用しているが，我々は，次の理由(1)，(2)より，解釈としてエルブラン解釈のみを考える多ソートの述語論理（ＭＨ１とよぶ）を採用する。

(1) 意味定義の対象を仕様記述に用いる自然語に限定しているので，内包については，プログラムの入力パラメータに関する限量を簡潔に表現する機能があれば十分であり，高階論理を用いる必要はない。

(2) 多ソートの概念は，プログラムや仕様における抽象的データタイプを定義するのに有用である。

さて，形式論理や代数的公理系などの形式的体系では，記号集合は，a)変数や量記号“∀”，“∃”のように，単一の式で一般に無限の式の集合を定義するために用いられる記号と，b)関数記号，述語記号のように，議論の対象となる式を直接構成する記号の集合に分割できる。形式論理では，ブール値を特別扱いしているので，b)はさらに，b1)論理記号とb2)非論理記号（関数記号と述語記号）に分割される。自然語においても，その語いは，a)限定詞（冠詞を含む），関係代名詞や指示代名詞のように，同一対象を指示したり対象の値の取り得る範囲を指定する機能をもつもの（メタ要素とよぶ）と，b)議論の対象，及び対象間の関係・演算を直接指示する語句とに分割される。b)はまた，論理の場合との類比により，b1)接続詞，助動詞等，ブール値上の演算に対応するもの（論理的要素）と，b2)一般の動詞や名詞等，対象領域毎に意味の定まるもの（非論理的要素）に分割できる。本手法では，次の(D1)～(D3)によって意味定義を行う。

(D1) 論理体系ＭＨ１の定義を行う。

(D2) メタ要素，論理的要素の意味を定義するため，L_{NS}からＭＨ１の論理式の集合L_{MH1}への変換則を定める。

(D3) 非論理的要素の意味を，対象領域毎に個別に定義する。

(D1)について，ＭＨ１では，量記号としては，存在記号“∃”，全称記号“∀”の他に，演算子“◇”を用いる。任意の論理式Ｐに対し，◇Ｐは，あるプログラム入力に対してＰが成り立つことを表す。なお，各述語記号及び関数記号にプログラム入力を表す引数を追加することにより，一般に“◇”を含む論理式（の集合）は，それを含まない論理式（の集合）で表現することができる。

次に，変換則の定義法の概略を述べる。自然語文，または自然語文における全部または一部の名詞句を変数で置き換えて得られる系列を，以下“擬似自然語文”とよぶ。特に，すべての名詞句を変数で置き換えて得られる系列を，“素擬似自然語文”とよぶ。同様に，名詞句において，部分句としての名詞句をすべて変数で置き換えて得られる系列を，“素擬似名詞句”とよぶ。ｓをを擬似自然語文，ｓ’を論理式とする。“ｓとｓ’が対応する”という述語“ｓ≈ｓ’”を，以下のように再帰的に定義する。

(1) s_0を素擬似自然語文とする．s_0の変数xに素擬似名詞句αを代入した式をs_1とし，s_0に対して，αで定まる変形をほどこして得られる式をs_1'とする（例えば，α＝a recordならば，$s_1' = \exists x[x \in record \wedge s_0]$）．このとき，$s_1 \approx s_1'$が成り立つ．

(2) $s_{i-1} \approx s_{i-1}'$が成り立つと仮定する．(1)と同様の操作によりs_{i-1}及びs_{i-1}'から得られる式をそれぞれs_i，s_i'とすると，$s_i \approx s_i'$が成り立つ．

(3) 上の(1)(2)以外の対応は認めない．□

$s \approx s'$が成り立ち，かつ，sが自然語文，すなわち変数を含まない擬似自然語文ならば，sはs'に変換されるといい，sの意味を，論理体系ＭＨ１におけるs'の意味と等しいと定義する．自然語文sに対し，$s \approx s'$かつ$s \approx s''$を満たす論理的に等価でない論理式s'，s"が存在する可能性がある．「各sに対し，$s \approx s'$を満たす論理式が（論理的に等価な範囲で）高々一つしか存在しない」という意味であいまいでない仕様を記述することは，一般には書き手の責任と考える．

なお，複数の文にわたって同一の対象を表す名詞句が現れる場合があることを考慮し，変換の単位は文の適当な系列，例えば段落とする．

３．構文則の定義

3.1 構文則の記述方針

我々が考えている英語の部分言語L_{NS}を，通常の文脈自由文法の記法で記述しようとすると，規則の数がかなり多くなり，文法全体の見通しが悪くなる．そこで，GPSG[2]を参考にして，非終端記号が引数を持つ形で構文則を記述する．ただし，引数がとりうる値の数を有限としたので，この文法は文脈自由文法に展開することができる．

また，GPSGは自然言語に対してより一般的な説明を与えようとして，複雑なマクロを用いているが，我々は対象を仕様記述に用いる英語の部分言語にしぼり，単純なマクロのみを用いて構文則を記述している．

3.2 構文則の記述法

文法の記述は，$A \rightarrow A_1, \cdots, A_n$という形で行う（A，$A_1, \cdots, A_n$は非終端記号）．各規則において右辺の非終端記号が一つ指定されており，それをHEADと呼ぶ．HEADは，“動詞句における動詞”等，句の核となる部分句を導出する非終端記号である．また，非終端記号は，$A\,[F_1\ fv_1, \cdots, F_{n_A}\ fv_{n_A}]$（$n_A$はAにより決まる整数）の形で記述する．Aをカテゴリ名と呼ぶ．例えば，S，VP，AP，NP，N，PP，DETは，それぞれ，文及び節，動詞句，形容詞句，名詞句，一般名詞句，前置詞句，冠詞を導出する非終端記号のカテゴリ名である．ここで，$F_1, \cdots, F_{n_A}$を引数，$fv_1, \cdots, fv_{n_A}$を（それぞれ，引数$F_1, \cdots, F_{n_A}$の）値と呼ぶ．各カテゴリ名がどのような引数をとるかは定まっており，また，各引数がとりうる値も有限個定まっている．これらの引数で規則の中で特に値が記述されていないものは，ある制約を満たす範囲で任意の値をとることができる．そしてその制約により，主語と述語の数・人称の一致，および関係節を作る際のいわゆるgap[2]の扱い等を，簡潔かつ自然に記述することができる．

3.3 構文則の拡張

3.3.1 拡張のねらい

我々がこれまでに用いていた構文則[8]は対象分野を限定しない一般的な英語の文法を記述したものであった．そのため，構文にはかなりのあいまい性がある上，語句のレベルでデータタイプを考慮した構文上の条件（例えばcontain（の一つの用法）は，あるデータタイプdに対し，dのある集合と，dのある要素を引数とする２項述語と定義される）と，それに基づく意味定義（語句の意味定義に関しては５．を参照）を代数的公理として与えており，構文の定義が二重構造になるという欠点があった．そこで各語句を抽象的データタイプの上の演算あるいは述語とみたときの構文上の情報（引数，および結果のデータタイプが何であるか）を非終端記号の引数として構文則に取りこめるようにする．また，通常の英文法記述のためにGPSGで用いられている制約に加えて，データタイプを記述する引数に関する制約を導入する．この方法をとることにより文法記述が繁雑になるのを回避できる．

3.3.2 拡張の概要

まず，二つのデータタイプd_1，d_2が“unifyする”という述語を定義しておく．例えば，d_1=set of record，d_2=set of d（dはデータタイプを表す変数）の場合，d_1とd_2はunifyでき，その結果はset of recordである．また，系列が集合に適当な演算を追加して得られるデータタイプとして定義されているとすれば，d_1=sequence of d（dは変数）とd_2=set of records はunifyでき，その結果はsequence of records となる．

そして，各非終端記号に引数"DATATYPE"を新たに付け加える．名詞を表わす非終端記号，N，NP，および，他の語の引数となる前置詞句PP（前置詞句には修飾語になるものと，引数になるものがあり，それは，PPの引数PFORMの値により区別できる[2]）などでは+d（dはfile，recordなどのタイプ名）であり，V，VPなど動詞を表わす非終端記号では$[-d_1, -d_2, \cdots, -d_k, +d, -d_{k+1}, \cdots, -d_n]$（d，$d_i$（$1 \leq i \leq n$）はタイプ名）である．$d_i$は直観的には動詞を抽象的データタイプ上の演算とみた場合の引数のデータタイプに，dは演算の結果得られる値のデータタイプに相当する．

動詞，名詞の各語句（終端記号）における引数の値は各語句毎に定める．

例えば次の文(3-1)で，file，containのDATATYPEの値はそれぞれ+file，[-file，+bool，-set of record]となる．

The file contains record. (3-1)

次に規則をi) HEADを関数記号，右辺にあるその他の非終端記号を引数とみたとき，関数適用とみなせる規則と，ii) DATATYPEの値がHEADと左辺の非終端記号で等くなる規則に分割する．i)の例としては，$S \rightarrow NP\ VP_H$，$VP \rightarrow V_H\ NP\ NP$（それぞれ，文，および第４文型の動詞句を生成する規則で，

添字Hのついた非終端記号がHEAD），ii)の例としてはN→AP N_H（名詞を形容詞で修飾する規則）がある．そしてi)，ii)のグループ中の各規則における非終端記号の引数DATATYPEのとりうる値に対して以下のような制約を設ける．

A→α H βをi)に属する任意の規則とする．ただし，HはHEAD，α，βは非終端記号列．α，β中の非終端記号の数をそれぞれ，|α|，|β|とする．α中の非終端記号のDATATYPEの値を$+n_1$，…，$+n_{|\alpha|}$，β中の値を$+m_1$，…，$+m_{|\beta|}$，HのDATATYPEの値を，$[-d_1, -d_2, \cdots, -d_k, +d, -d_{k+1}, \cdots, -d_n]$とする．このとき，$d_{k-|\alpha|+1}$，…，$d_k$と$n_1$，…，$n_{|\alpha|}$，および，$d_{k+1}$，…，$d_{k+|\beta|}$と$m_1$，…，$m_{|\beta|}$は，それぞれunifyしなければならない．

そして左辺の非終端記号のDATATYPEの値は$[-d_1, \cdots, -d_n]$からunifyしたものを除いたもの，すなわち$[-d_1, \cdots, -d_{k-|\alpha|}, +d, -d_{k+|\beta|+1}, \cdots, -d_n]$とする．

ii)に属する規則の左辺の非終端記号とHEADではDATATYPEの値は一致しなければならない．

このように構文則を拡張することにより，構文のあいまい性が減少するだけでなく，各非終端記号のデータタイプがわかるので変換(4.参照)の際のあいまいさも小さくなる．また，拡張は引数一つと，それが満たすべき制約を加えるという形で行えるので，当初我々が定義した規則の修正は少なくてすむ．

現在，終端記号を導出する規則を除き，87の規則がある．

4．変換則の定義

2.で述べた方針に従い，変換則，すなわち，述語“≈”を定義する公理を与える．まず，4.1で変換則の概要について説明し，4.2，4.3でそれぞれ複数名詞句を含む文および，いわゆるdonkey文の意味定義を考慮した一般的な変換則について述べる．4.4では等位接続の意味定義について述べる．

4.1 変換則の概要

本稿での定義法では，自然語文sを変換した結果得られる論理式s'において異なる変数が存在する場合，それらの変数の有効範囲は，sの構文によって定まるとする．そこで，変換則を定義する際，sに現れる変数xへの素擬似名詞句の代入は，sに，xよりも“有効範囲の狭い”変数が存在しないときのみ許すように定義する．一般に，式sの変数xに代入可能であることを表す述語を，ok_sbst_for x in sで表す．ただし，擬似自然語文中の着目する変数のうち，最も左側のものが従属節（例えば関係節やif節）に含まれている場合（例えば，If the file contains x, x should be printed. において変数 xに代入を行う場合）は，以下で述べる変換則は適用できないとする．このような場合も考慮して拡張された変換則については4.3で述べる．

また，変数xと素擬似名詞句αのデータタイプが一致するという述語をtype_check(x, α)で表わす．

以下，ASL／＊の公理を用いて変換則を記述する．

ℓ_1==r_1, ℓ_2==r_2, …, ℓ_m==r_m >> L == R

で条件付き公理を表す．特に条件部がない場合は L == R とかく．簡単のため，条件部が等しいn個の公理を一つにまとめて，

ℓ_1==r_1, …, ℓ_m==r_m >> L_1 == R_1, …, L_n==R_n

とかく．また，ℓ == true の形の公理の条件部または本体を，ℓと略記することがある．公理の変数は以下のとおり．

(1) x, y, …, X, Y, …等は擬似自然語文や論理式自身に現れる変数を表す．

(2) α，β，…等は素擬似名詞句を構成する句を表す変数．αに対応する非終端記号がNであるときα: Nと記述する．

(3) s, s_1, s_2, …は擬似自然語文を表す．

(4) s', s'_1, s'_2, …は論理式を表す．

[変換則T1] α: N[-PLU]（一般名詞単数形，-PLUは単数を+PLUは複数を表わす）

β: NP[+POSS]（+POSSは所有格を表す）

s≈s', ok_sbst_for x in s >>

[1a] s{{x/a α}} ≈
　if GEN(s, x) then ∀x[x∈α <=> s']

[1b] s{{x/a α}} ≈
　if EXIST(s, x) then ∃x[x∈α ∧ s']

[2] s{{x/every α}} ≈ ∀x[x∈α ⊃ s']

[3] s{{x/no α}} ≈ ¬∃x[x∈α ∧ s']

[4] s{{x/the α}} ≈ s'{x/α}

[5] s{{x/β α}} ≈ s'{x/β α}

[T2] α: NP[-PLU, +TYPE]（+TYPEは抽象名詞を表す）

β: PPNAME（固有名詞）

s≈s', ok_sbst_for x in s >>

s{{x/α}} ≈ s'{x/α}

s{{x/β}} ≈ s'{x/β} □

ここで，s{{x/α}}は，式sの最初に現れるxにαを，それ以降のxに適当な指示代名詞を代入して得られる式．s'{x/α}は，s'の全てのxにαを代入して得られる式を表す．s{{x/α}}，s'{x/α}のASL／＊による定義は省略する．[T1]の[1a]において，GEN(s, x)は，擬似自然語文sにおいて，xにa αの形の名詞句を代入したとき，それが，‘generic’な意味をもつ（全称記号に対応する）ときtrueとなる述語で，[1b]のEXIST(s, x)は，それが存在記号に対応するときtrueとなる述語である．GEN(s, x)，EXIST(s, x)の定義は，s中の動詞句を構文的に分類すること等によって定める．詳細は検討中である．[T1]の[1]は，擬似自然語文sと論理式s'が対応することが既に知られており，かつsのxに擬似名詞句αの代入を行えるとき，GEN(s, x)がtrueならば，s{{x/a α}}は∀x[x∈α <=> s']と対応し，EXIST(s, α)がtrueのとき，∃x[x∈α ∧ s']と対応することを表す．[T1]の[4]，[5]及び[T2]は，the を冠詞として持つ名詞句，所有格を伴う名詞句，抽象名詞，固有名詞については，論理式においても，これらを“項”とみなして代入すること等を表す．限定詞 some 及び形容詞句 at least one 等については上の[1b]と，

each, any については[2]と同様に定義する．

〔例〕s_1=x should be updatedは，素擬似自然語文であるから，s_1はs_1自身に対応する．

$s_2=s_1\{\{x/\text{every record of y}\}\}$

=every record of y should be updated　は

$s_2' = \forall x[x \in \text{record of y} \supset x \text{ should be updated}]$

に対応する．　([T1]の[2]より)

$s_2\{\{y/\text{a file}\}\}$

=every record of a file should be updated　は

$\exists y[y \in \text{file} \wedge \forall x[x \in \text{record of y} \supset$

x should be updated]]

に対応する．　([T1]の[1b]より)□

上の例で，“is”は非論理的要素とみなしている．

以上の変換則では，擬似名詞句に現れる関係節や分詞による修飾句はそのまま対応する論理式中に現れる．以下では，関係節に関する変換則を与える．

[T3] α: N, β: S[WH [R]]

norm(x, β)$\approx$s' , x is_not_in α, x is_not_in β

>>

$x \in \alpha\ \beta \iff x \in \alpha \wedge s'$　□

norm(x, β)は，関係節β中，先行詞の抜けた場所に変数xを補い，関係代名詞を含む句が文頭に移動している場合には，それをもとの位置に戻して得られる文．norm(x, β)の定義は省略するが，非終端記号の引数の値を利用することにより定義できる．

〔例〕x is an integer which is less than 10 は

$\exists y[y \in$ integer which is less than 10 $\wedge$ x is y]

に対応する．　([T1]より)

上の式中の句 y$\in$integer which is less than 10 は

y$\in$integer $\wedge$ y is less than 10

に対応する．　([T3]より)□

その他，分詞による修飾，接続詞で結ばれた動詞句，同格などの変換則も定義されている．

4.2　複数名詞句に関する変換則

複数名詞が，対応する論理式においてどのような機能を果すかを考える．

The file contains customer records.

Count the number of them.　(4-1)

The file contains some customer records.　(4-2)

The file contains no customer records.　(4-3)

例文(4-1)ではcustomer recordsをさす代名詞themは，the fileに含まれているcustomer recordの集合を表していると考えられる．このように，無冠詞の複数名詞句は，文脈およびその語の意味によって決まる集合を表すとする．また，限定詞＋複数名詞の形の名詞句をもつ例文(4-2)，(4-3)はそれぞれ，the fileに含まれるcustomer recordのある空でない集合が存在すること，および存在しないことを表していると考えられる．

一方，次の例文，

All processes print their current states.　(4-4)

では，各プロセスが自分自身の状態をプリントするという意味であり，代名詞theirはプロセスの集合ではなく個々のプロセスを受けていると考えられる．したがって，対応する論理式において，theirはプロセスの集合をさす変数ではなく，個々のプロセスをさす変数に対応しなければならない．すなわち，代名詞の扱いが(4-1)～(4-3)と(4-4)では異なる．

変換則はこれらの文に対応できるよう定めた．以下に複数名詞に関する変換則を示す．

[T4] s$\approx$s' , ok_sbst_for x in s,

x belong_to X in s >> s[[x/X]] $\approx \forall x \in X \supset s'$

[T5] α: N[+PLU]

s$\approx$s' , ok_sbst_X in s, type_check(X, α) >>

[1a] s{{X/α}} $\approx$

if GEN(s, X) then s' {X/α'}

[1b] s{{X/α}} $\approx$

if EXIST(s, X) then $\exists X[X \in$ nonempty set of α' $\wedge s']$

[2] s{{X/some α}} $\approx \exists X[X \in$ nonempty set of α' $\wedge s']$

[3] s{{X/no α}} $\approx \neg \exists X[X \in$ nonempty set of α' $\wedge s']$

ここで，α'はαの単数形を表しているが，もしαが単数形のない名詞の場合は，nonempty set of α'の代りに，αとかく．述語 x belong_to X in s は式sにおいて，xのデータタイプをdとすると，Xのデータタイプがset of dであるとき，かつそのときのみtrueとなる．また，s[[x/X]]は，変数xをXに置き換え，xが主語のときには動詞の形を，所有格のときにはそれが付く名詞を複数にする．(主語につく所有格のときは，その両方を行う．)

Some processes print their states.　(4-5a)

Detect those processes.　(4-5b)

先程と同様，(4-5a)のtheirは個々のプロセスを表しており，対応する論理式には個々のプロセスをさす変数が必要である．しかしながら，(4-5b)のthose processesは(4-5a)で言及されたプロセスの集合を表しており，対応する論理式には(4-5a)で言及されたプロセスの集合を表す変数を用いなければならない．[T4]により，個々の元を表す変数（次の例ではp）と，それを要素にもつ集合を表す変数（次の例ではP）との対応をつけることができる．[T5]は集合を表す変数Xに無冠詞の複数名詞，および限定詞some, noのついた複数名詞を代入するときの規則である．このときαは複数形なので，type_check(X, α)という条件より変数Xは集合型となる．

〔例〕p prints s. Detect P.

$\approx$ p prints s $\wedge$ the program detects P(※)

(※命令文における主語の省略については，文献(7) (8)を参照．また，複数の素擬似自然語文は，それらを論理記号“$\wedge$”で結んだ論理式に対応する．

[T1] [5]より

p prints p's state. Detect P.

$\approx$ p prints p's state $\wedge$ the program detects P

p belong_to P in s == true　なら，[T4]より
P print P's states. Detect P.
≈ ∀p∈P ⊃ [p prints p's state ∧
　　the program detects P]
[T5] [2]より
Some processes print their states.
Detect those processes.
≈ ∃P∈nonempty set of process ∧
　[∀p∈P ⊃ [p prints p's state ∧
　　the program detects P]
〔例〕Xのデータタイプは，xのデータタイプの集合型であるとする．
[T4]より，
X are read. ≈ ∀x∈X⊃ x is read
GEN(X are read, X)がX-cardsに対してtrueであるとすれば，[T5] [1]より
cards are read. ≈ ∀x∈card⊃x is read
となる．
〔例〕f contains CR. p counts the number of CR.
は，f contains CR∧p counts the number of CRに対応する．
[T5]より
f contains customer records.
p counts the number of them.
≈ ∃CR∈nonempty set of customer record ∧
　f contains CR∧p counts the number of CR
[T1]より
the file contains customer records.
Count the number of them.
≈ ∃CR∈nonempty set of customer record ∧
　the file contains CR∧the program counts
　　the number of CR

なお，上の例で語句"contain"の意味は，その二つの引数（主語と目的語）のデータタイプが順にset of d，及びdである場合の定義を拡張して，引数のデータタイプがともにset of dである述語として次のように定義する．
∀D_1, D_2∈set of d
　D_1 contains D_2 <=> ∀e∈D_2 ⊃ D_1 contains e
fileがset of recordに演算を追加して得られるデータタイプとすれば，上の公理より，
∀f∈file, ∀R∈set of record
　f contains R
<=> ∀r∈R ⊃ f contains r

4.3 donkey文に関する変換を考慮した変換則

一般にdonkey文と呼ばれる文があり，例えば，
If the file contains a record, it should be printed.　(4-6)
という文がそれである．if節中の“a record”の“a”は存在作用素に対応するように思われるが，実際にはこの文は，「“the file”に含まれるようなrecordは必ずprintされる」と解釈されるから，対応する論理式は，
∀x [[x∈record ∧ the file contains x] ⊃
　x should be printed]　(4-7)
となり，xを限量するのは全称記号である．このようなdonkey文は，代名詞や関係節を含む複文にみられる．以下で，donkey文の解析とそれに基づく変換則を述べる．まず，次の擬似自然語文(4-8)を考える．
If the file contains x, x should be printed.　(4-8)
(4-8)に対応する論理式は，
the file contains x ⊃ x should be printed　(4-9)
である．
(4-6)は(4-8)の変数xに“a record”を代入して得られる．このとき，(4-6)に対応する論理式(4-7)は，(4-9)から次のようにして定義されると考える：まず(4-9)において，(4-8)の従属節“If the file contains x”と対応する部分論理式の先頭にxを限量する存在記号をつけた次の“中間的な論理式”を考える．
∃x [x∈record ∧ the file contains x] ⊃
　x should be printed
=¬ [∃x [x∈record ∧ the file contains x]] ∨
　x should be printed　(4-10)
を考える．(4-10)で，すべての自由変数xを含むように，存在記号を否定“¬”の外側に移動すると(4-7)が得られる．

このように，論理式では限量すべき変数すべてを含むように量記号の位置を決めるのに対し，自然語では局所的に決まる量記号に基づいて冠詞の種類が決まるため，donkey文のような現象が起こると考えられる．そこでdonkey文に対する変換を考慮し，以下のように変換則を拡張する．
[変換則T1'] α: N[-PLU]
s ≈ s', ok_sbst_for x in s　>>
[1a] s{{x/a α}} ≈
　if GEN(s, x)　then $Q_\forall$(s', x, α, s)
[1b] s{{x/a α}} ≈
　if EXIST(s, x) then $Q_\exists$(s', x, α, s) □
ここで，$Q_\forall$(s', x, α, s)は次のように定義される．
(1) sにおける最初の（一番左側）の変数xを含む節に対応するs'の部分論理式をs'_1とする．s'においてs'_1をs''_1 = ∀x[x∈α <=> s'_1]で置換えた式をs"とする．
(2) s"において，s''_1の外側に現れる自由変数xをすべて含む位置に，s''_1の先頭の量記号“∀”を移動させた（ただし，“¬”あるいは“⊃”の外側に出るたびに“∀”は“∃”，“∃”は“∀”に置換える）結果得られる式を$Q_\forall$(s', x, α, s)とする．□
$Q_\exists$(s', x, α, s)についても同様に定義する．4.1，4.2で述べた他の変換則もすべて上記のように変更する．

なお，上の$Q_\forall$(s', x, α, s)の定義において，s"を作るためには，量記号をつけるべき部分論理式が指定できなければならないが，それは，擬似自然語文sにおいて一番左側に現れるxに対応する変数を含む部分論理式である．

擬似自然語文が複数の同一変数を含む場合を考慮し，同一変数には（仮に）添字をつけ，一つの式内で同一変数の添字がすべて異なるようにしておけば，擬似自然語文の各出現における変数が，論理式においてどの出現の変数に対応するかがわかる。

4.4 等位接続された名詞句

次のような文を考える。

x, y and z are integers. (4-11)

これは，

x is an integer and y is an integer and
z is an integer. (4-12)

という文と意味の上で等価である。(4-12)における“and”は論理記号“∧”の意味を持つので，(4-11)の文の意味は(4-12)の各文（“x is an integer”等）を論理式に変換し，それらを論理記号“∧”で結合して得られる論理式によって定義できる。このように名詞句を列挙する等位接続（例えば“A, B and C”，“A, B or C”）の構文には，等位接続の部分を各名詞句（“A”，“B”，“C”）で置換えた構文を論理的に（∧，∨で）結合したものとして定義されるものがある。これを等位接続の展開と呼ぶことにする。一方，

A stream of telegrams is available as a sequence of letters, digits and blanks. (4-13)

という文は，

A stream of telegrams is available as a sequence of letters and
a stream of telegrams is available as a sequence of digits and
a stream of telegrams is available as a sequence of blanks. (4-14)

という文の意味と等価ではなく，従って展開できない。これは，(4-13)の文では，“and”は(4-11)の文と同様に集合和をとる演算子ではあるが，等位接続が“a sequence of”の目的語となっているために(4-11)の文のように論理結合子“∧”の意味をもたないためである。

このように等位接続が展開できるかどうかは，等位接続が主語，目的語，補語となっているところの動詞や前置詞に固有の性質によって決まる。従って，自然語文を論理式に変換する際に等位接続された名詞句があれば，等位接続を含む語句に関する情報を参照し，等位接続の展開が可能な限り，順次展開を行う。

5. 論理式への変換と語句の定義の例

ここでは，文献(3)の例題を一つ取りあげ，実際にそれを紙上で変換した結果と，その中に現れている語句の定義を示す。表1に変換の対象となる英文を示す。[N1]において(3)のthese two filesは(1)のa serial master file および(2)のanother serial file を受けているので，これらを一つの段落とみなし，まとめて変換する。[N2]は助動詞“may”を含んでおり，プログラムへの入力によっては，invoiceを2つ以上もつcustomerが存在することを述べて

表1 変換の対象となる自然語仕様

[N1] (1) A serial master file contains customer name and address records, arranged in ascending sequence by customer number.

(2) Another serial file contains billable item records, arranged in ascending sequence by date within invoice number within customer number.

(3) These two files are to be used to print invoices.

[N2] There may be more than one invoice for a customer but some customers will have no invoices.

[N3] Due to punching errors there may be billable item records for which no customer name and address record exists; these are to be listed on a diagnostic file of messages.

表2 表1から変換の結果得られる論理式

[L1] ∃x[x∈ serial master file ∧
∃CR[CR∈nonempty set of customer
name and address record ∧
CR are arranged in ascending seqenece
by customer number ∧
x contains CR] ∧
∃z[z∈serial file ∧
∃BR[BR∈nonempty set of
billable item record ∧
BR are arranged
in ascending sequence by date
within invoice number
within customer number ∧
z contains BR] ∧
∃I[I∈nonempty set of invoice ∧
{x, z} are to be used to print I]]]

[L2] ◇∃x[x∈customer
∃I[I∈more than one invoice(*1) ∧
∀i[i∈I ⊃ i exists for x]] ∧
∃C[C∈nonempty set of customer ∧
¬∃I[I∈nonempty set of invoice ∧
C will have I]]

[L3] ◇∃E[E∈nonempty set of punching error ∧
∃BR[BR∈nonempty set of billable item record ∧
¬∃CR[CR∈nonempty set of customer name
and address record ∧
due to E, CR exist for BR] ∧
∃f[f∈diagnostic file of messages ∧
BR are to be listed on f]]]

(*1) 名詞句 “more than one α”（αは単数形の普通名詞）は，複数形であり，対応する非終端記号はNP[+PLU]。

いる．対応する論理式は，“may”に関する次の変換則を用いて導いた．

[変換則T6] s : S[+Kn, +MAY], $s \approx \Diamond \bar{s}$ □

ここで，+Knは，素擬似自然語文であること，+MAYは，主節が助動詞“may”をもつことを表す．$\bar{s}$は，sを平叙文に変形した素擬似自然語文．

変換の結果得られる論理式を表2に示す．[L2]，[L3]を導く際に，語句“there”に関する以下の公理を用いた．

∀ d ∈data type, ∀x, y∈d, ∀X∈set of d

there is x -- true

there are X for y -- ∀x∈X ⊃ x exists for y

また，表1で用いられている語句“record”に関する定義を表3に示す．[L2]の“within”句を含む論理式中の語句“x is arranged in ascending sequence by y within z within w”は，表3のii)と同様に定義できる．

謝辞　文献(2)等について御教示頂いた，本学言語文化部の郡司　隆男助教授に深謝します．

文　献

(1) D. R. Dowty, R. E. Wall and S. Peters: “Introduction to Montague Semantics”, D. Reidel Publishing Company (1981).

(2) G. Gazdar, E. Klein, G. Pullum and I. Sag: “Generalized Phrase Structure Grammar”, Basil Blackwell (1985).

(3) M. A. Jackson: “Principles of Program Design”, pp. 49-50, Academic Press (1975).

(4) 石木，並河，関，杉山，藤井，鳥居，中小路：“プログラム仕様に用いる自然語の処理システム　－代数的仕様への変換と知識管理－”，情報処理学会研報，SW-52-6 (1987-2).

(5) 嵩，谷口，杉山，関：“代数的言語ＡＳＬ／＊　－意味定義を中心に－”，信学論(D)，Vol. J69-D, No. 7, pp. 1066-1074 (1986-7).

表3　record型の語句に関する定義

i) ∀r∈record, ∀d∈data type

d of r∈data type

ii) ∀y∈data type(*1), ∀r∈nonempty set of record, ∀x∈sequence of r, ∀i, j∈int,

x is arranged in ascending sequence by y∈bool ∧

x is arranged in ascending sequence by y

<=>[i<j ⊃ y of i_th x < y of j_th x](*2) -- true

iii) ∀d∈data type, ∀e∈nonempty set of d,

more than one d∈nonempty set of d ∧

e∈more than one d <=> the_number_of e ≧ 2(*3)

-- true

(*1) ただし，yは大小関係が定義されているデータタイプ

(*2)，(*3) 系列に関する演算i_th，集合に関する演算the_number_ofのＡＳＬ／＊による定義は，文献(7)参照

(6) 白井　賢一郎：“形式意味論入門”，産業図書 (1985).

(7) 関，並河，藤井，嵩：“自然語によるプログラム仕様の形式的意味定義　－自然語による仕様から代数的仕様への変換－”情報処理学会研報，SW-52-7 (1987-2).

(8) 並河，関，藤井，嵩：“プログラム仕様記述に用いる自然語の形式的意味定義”，ＬＡシンポジウム (1986-7).